



**UNIVERSIDAD TECNOLÓGICA  
NACIONAL  
FACULTAD REGIONAL  
RESISTENCIA**

## **INGENIERÍA EN SISTEMAS DE INFORMACIÓN**

**MATERIA:** SISTEMAS DE GESTIÓN DE BASES DE DATOS

**NIVEL:** 3

**CICLO LECTIVO:** 2024 – 2DO. CUATRIMESTRE

**DOCENTES:**

PROFESOR: I.S.I. ANDRÉS PABLO FANTÍN

J.T.P.: I.S.I. JUAN CARLOS FERNÁNDEZ

AUXILIAR ADSCRIPTA: LUCIANA CAMPESTRINI

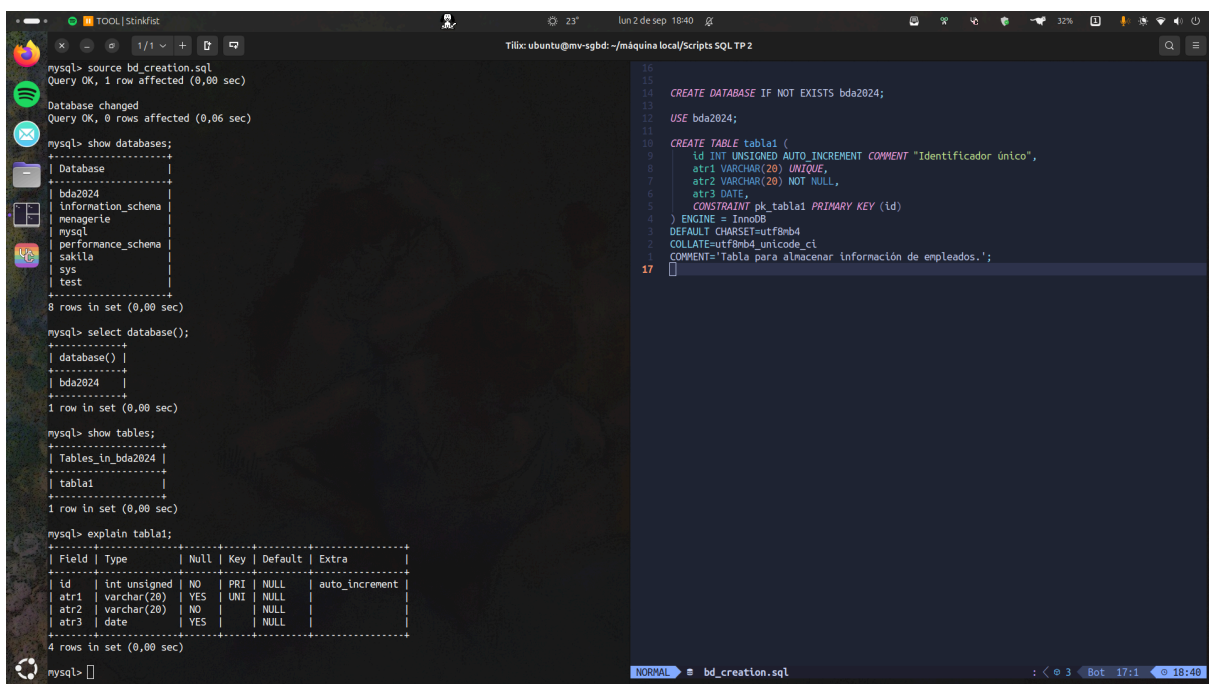
**Grupo n.º2:**

- Arduña Zago, Agustín Juan Luis
- Berón de Astrada, Santiago Agustín
- Fernández, Facundo Nahuel
- Kinweiler, Víctor Fabricio
- Schefer, Mauricio Nicolás
- Velazco Gez Schegtél, Juan Ignacio

# Trabajo Práctico N°2

**Consigna:** Crear una base de datos con el año actual como nombre y dentro de la misma una tabla con distintos campos.

- CREATE DATABASE bda2019;
- USE bda2019;
- CREATE TABLE tabla1(  
id int, campo1 VARCHAR(20),  
PRIMARY KEY (id));



```
mysql> source bd_creation.sql
Query OK, 1 row affected (0,00 sec)

Database changed
Query OK, 0 rows affected (0,06 sec)

mysql> show databases;
+-----+
| Database |
+-----+
| bda2024   |
| information_schema |
| menagerie |
| mysql     |
| performance_schema |
| sakila    |
| sys       |
| test      |
+-----+
8 rows in set (0,00 sec)

mysql> select database();
+-----+
| database() |
+-----+
| bda2024    |
+-----+
1 row in set (0,00 sec)

mysql> show tables;
+-----+
| Tables_in_bda2024 |
+-----+
| tabla1             |
+-----+
1 row in set (0,00 sec)

mysql> explain tabla1;
+-----+
| Field | Type          | Null | Key | Default | Extra          |
+-----+
| id     | int unsigned  | NO   | PRI | NULL    | auto_increment |
| atr1   | varchar(20)   | YES  | UNI | NULL    |                |
| atr2   | varchar(20)   | NO   |     | NULL    |                |
| atr3   | date          | YES  |     | NULL    |                |
+-----+
4 rows in set (0,00 sec)

mysql>
```

```
15
14 CREATE DATABASE IF NOT EXISTS bda2024;
13
12 USE bda2024;
11
10 CREATE TABLE tabla1 (
9     id INT UNSIGNED AUTO_INCREMENT COMMENT "Identificador único",
8     atr1 VARCHAR(20) UNIQUE,
7     atr2 VARCHAR(20) NOT NULL,
6     atr3 DATE,
5     CONSTRAINT pk_tabla1 PRIMARY KEY (id)
4 ) ENGINE = InnoDB
3 DEFAULT CHARSET=utf8mb4
2 COLLATE=utf8mb4_unicode_ci
1 COMMENT='Tabla para almacenar información de empleados.';
16
```

```
mysql> show databases;
+-----+
| Database |
+-----+
| bda2024  |
| information_schema |
| menagerie |
| mysql    |
| performance_schema |
| sakila    |
| sys      |
| test     |
+-----+
8 rows in set (0,00 sec)
```

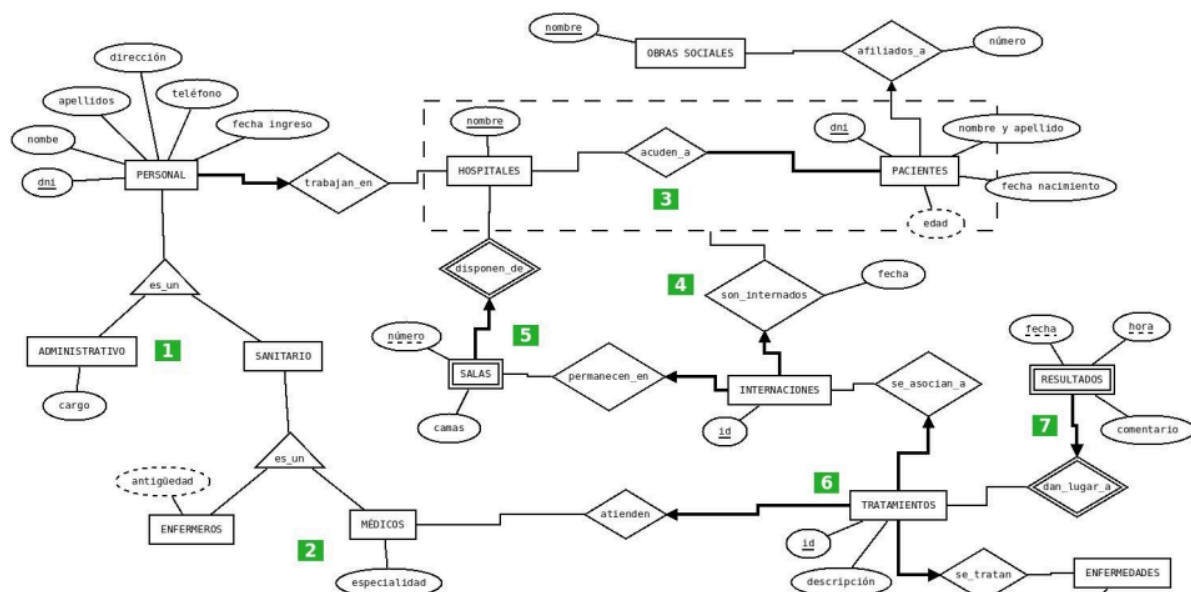
```
mysql> select database();
+-----+
| database() |
+-----+
| bda2024    |
+-----+
1 row in set (0,00 sec)
```

```
mysql> show tables;
+-----+
| Tables_in_bda2024 |
+-----+
| tabla1             |
+-----+
1 row in set (0,00 sec)
```

```
mysql> explain tabla1;
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| id    | int unsigned | NO   | PRI | NULL    | auto_increment |
| atr1  | varchar(20)  | YES  | UNI | NULL    |                |
| atr2  | varchar(20)  | NO   |     | NULL    |                |
| atr3  | date         | YES  |     | NULL    |                |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0,00 sec)
```

### Ejercicio 1: Consorcio de hospitales

Dado el siguiente enunciado y DER resultante:



CONSIDERACIONES:  
 - Se considera que en conjunto de entidades MÉDICOS no variará mucho en el tiempo, por lo que se instanciarán una sola vez y en ese caso será más adecuado plantear la especialidad como atributo en vez de entidad.  
 - Se considera paciente a una persona que desde la primera vez que acude a un hospital por cualquier tipo de consulta.  
 - De la atención de un paciente en un hospital puede surgir la necesidad de internarlo.  
 - Se debe generar un atributo "id" para identificar los tratamientos y las internaciones.

Se trata de diseñar la base de datos para la administración de un consorcio de hospitales, que permita gestionar datos acerca del personal así como de los pacientes de los mismos. De cada hospital interesa almacenar, además de su nombre dirección, teléfono, fax, etc., lo siguiente:

1. El personal de los hospitales (del que interesa almacenar su dni, nombre, apellido, dirección, teléfono y fecha de ingreso a trabajar al hospital) se divide en personal administrativo (se registra su cargo) y personal sanitario, dentro del que se distingue a su vez enfermeros (se registra su antigüedad en años) y médicos.
2. Los médicos tienen una especialidad que interesa conocer (pediatría, obstetricia, etc.) y sólo trabajan, al igual que el resto del personal, en un hospital.
3. Los pacientes pueden acudir a varios hospitales del consorcio.
4. Se desea conocer los datos personales de los pacientes que van a ingresar en el hospital, así como su obra social, número de afiliado, la fecha de admisión y la sala (habitación) en la que deben permanecer en caso de ser internados.
5. Cada sala se identifica por un número de sala dentro de cada hospital y se desea conocer el número de camas de las que dispone cada sala.
6. Cada admisión de un paciente en el hospital lleva asociada una o varias fichas de tratamiento en las que se indica la enfermedad y el médico que la atiende.
7. Cada tratamiento da lugar a distintos resultados que permiten realizar el seguimiento de cada enfermedad de un paciente. El resultado debe indicar la fecha y hora en que éste tuvo lugar, así como un comentario (por ejemplo, indicando si el paciente tiene fiebre etc.). Para un mismo tratamiento sólo puede haber un resultado en un mismo día, a una misma hora.

### Punto 1 :

Describir el proceso realizado para pasar al esquema relacional de la base de datos

correspondiente, detallando decisiones de diseño tomadas durante el mismo. Si hay elementos que aún no sabe cómo o no tiene herramientas para implementarlos (p.e. algunas restricciones),escríbalo.

Para realizar el esquema relacional de la base de datos del diagrama de entidad-relación, se utilizaron los nombres de cada una de las entidades para representar una respectiva tabla, así como también cada uno de sus atributos fueron definidos con sus tipos de datos que correspondan (como puede ser CHAR, INT, DATE, etc.). Luego de identificar las entidades, verificamos qué restricciones de participación tenía cada una de ellas, para determinar qué tipo de relación no es necesario su esquematización.

Además, para cada una de las entidades, se representan sus claves primarias y foráneas dentro del esquema.

Para las entidades débiles y de herencia tuvimos en cuenta utilizar las restricciones para asegurar que la información de la respectiva entidad se elimine en caso de eliminar la tupla correspondiente.

## Punto 2:

Realizar un diccionario de datos que contenga información para cada tabla de la siguiente manera:

PERSONAL				
NOMBRE DE LA COLUMNA	TIPO DE COLUMNA	ATRIBUTOS/ OPCIONES	VALOR PREDETERMINADO	DESCRIPCIÓN/COMENTARIOS
dni	INT	PK	NULL	Código identificador del personal.
nombre	VARCHAR(50)	-	NULL	Nombre del personal.
apellidos	VARCHAR(50)	-	NULL	Apellido del personal.
dirección	VARCHAR(50)	-	NULL	Dirección donde reside el personal.
teléfono	INT	-	NULL	Teléfono del personal.
fecha_ingreso	DATE	-	NULL	Fecha de ingreso del personal.
nombre_hospital	VARCHAR(50)	FK	NULL	Nombre del personal en el que trabaja el personal.
ADMINISTRATIVO				
dni	INT	PK, FK	NULL	Código identificador del personal administrativo.
cargo	VARCHAR(50)	-	NULL	Cargo particular del personal administrativo.
SANITARIO				

dni	INT	PK, FK	NULL	Código identificador del personal sanitario.
<b>ENFERMEROS</b>				
dni	INT	PK, FK	NULL	Código identificador del personal sanitario de enfermería.
antigüedad	DATE	-	NULL	Antigüedad del personal sanitario de enfermería.
<b>MEDICOS</b>				
dni	INT	PK, FK	NULL	Código identificador del personal sanitario médico.
especialidad	VARCHAR(100)		NULL	Especialidad del personal sanitario médico (pediatría, obstetricia, etc).
<b>TRATAMIENTOS</b>				
id	INT	PK	NULL	Código identificador del tratamiento.
descripción	TEXT	-	NULL	Descripción del tratamiento.
nombre_enfermedad	VARCHAR(50)	FK, NOT NULL	NULL	Nombre de la enfermedad que debe ser tratada.
dni_med	INT	FK, NOT NULL	NULL	Código identificador del médico encargado del tratamiento.
id_internación	INT	FK, NOT NULL	NULL	Código identificador de la internación del tratamiento.
<b>RESULTADOS</b>				
id	INT	PK	NULL	Código identificador de los resultados.
fecha	DATE	PK	NULL	Fecha de los resultados.
hora	TIME	PK	NULL	Hora de los resultados.
comentario	TEXT	-	NULL	Comentario acerca de los resultados.
<b>ENFERMEDADES</b>				
nombre	VARCHAR(50)	PK	NULL	Nombre de la enfermedad.
<b>INTERNACIONES</b>				
id	INT	PK	NULL	Código identificador de la internación.
numero_sala	INT	FK, NOT NULL	NULL	Número de la sala de internación.
nombre_sala	VARCHAR(50)	FK, NOT NULL	NULL	Nombre de la sala de internación.
fecha	DATE	NOT NULL	NULL	Fecha de la internación.
nombre_hospital	VARCHAR(50)	FK, NOT NULL	NULL	Nombre del hospital relacionado a la internación.

dni_paciente	INT	FK, NOT NULL	NULL	DNI del paciente internado.
<b>HOSPITALES</b>				
nombre	VARCHAR(50)	PK	NULL	Nombre del hospital.
<b>SALAS</b>				
nombre	VARCHAR(50)	PK,FK	NULL	Nombre de la sala.
número	INT	PK	NULL	Número de la sala.
camas	SMALLINT	-	NULL	Número de la cama de la sala
<b>PACIENTES</b>				
dni	INT	PK	NULL	Código identificador del paciente.
nombre_apellido	VARCHAR(50)	-	NULL	Nombre y apellido del paciente.
fecha_nacimiento	DATE	-	NULL	Fecha de nacimiento del paciente.
nombre_obra_social	VARCHAR(50)	FK	NULL	Nombre de la obra social a la que está afiliado el paciente.
número	INT	NOT NULL	NULL	Número de afiliado del paciente.
edad	INT	-	NULL	Edad del paciente.
<b>ACUDEN_A</b>				
nombre	VARCHAR(50)	PK, FK	NULL	Nombre del hospital al que acude el paciente.
dni	INT	PK, FK	NULL	Código del paciente que acude al hospital.
<b>OBRAS SOCIALES</b>				
nombre	VARCHAR(50)	PK	NULL	Nombre de la obra social

### Punto 3:

Escribir el script SQL con las sentencias de creación de los objetos en el motor de base de datos y mostrar los resultados de su ejecución.

### **Scripts SQL**

```
CREATE DATABASE IF NOT EXISTS hospital;
USE hospital;
```

```
CREATE TABLE personal (
```

```

    dni INT,
    nombre VARCHAR(50),
    apellidos VARCHAR(50),
    direccion VARCHAR(50),
    telefono INT,
    fecha_ingreso DATE,
    CONSTRAINT pk_personal PRIMARY KEY (dni)
);

CREATE TABLE administrativo (
    dni INT,
    cargo VARCHAR(50),
    CONSTRAINT pk_administrativo PRIMARY KEY (dni),
    CONSTRAINT fk_administrativo FOREIGN KEY (dni) REFERENCES
personal(dni) ON DELETE CASCADE
);

CREATE TABLE sanitario (
    dni INT,
    CONSTRAINT pk_sanitario PRIMARY KEY (dni),
    CONSTRAINT fk_sanitario FOREIGN KEY (dni) REFERENCES personal(dni) ON
DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE enfermeros (
    dni INT,
    antiguedad DATE,
    CONSTRAINT pk_enfermeros PRIMARY KEY (dni),
    CONSTRAINT fk_enfermeros FOREIGN KEY (dni) REFERENCES sanitario(dni)
ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE medicos (
    dni INT,
    especialidad VARCHAR(100),
    CONSTRAINT pk_medicos PRIMARY KEY (dni),
    CONSTRAINT fk_medicos FOREIGN KEY (dni) REFERENCES sanitario(dni) ON
DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE tratamientos (
    id INT,
    descripcion TEXT,
    CONSTRAINT pk_tratamientos PRIMARY KEY (id)
);

```



```
CREATE TABLE resultados (  
    id INT,  
    fecha DATE,  
    hora TIME,  
    comentario TEXT,  
    CONSTRAINT pk_resultado PRIMARY KEY (id, fecha, hora),  
    CONSTRAINT fk_resultado FOREIGN KEY (id) REFERENCES tratamientos(id)  
ON DELETE CASCADE ON UPDATE CASCADE  
);
```

```
CREATE TABLE enfermedades (  
    nombre VARCHAR(50),  
    CONSTRAINT pk_enfermedades PRIMARY KEY (nombre)  
);
```

```
CREATE TABLE internaciones (  
    id INT,  
    CONSTRAINT pk_internaciones PRIMARY KEY (id)  
);
```

```
CREATE TABLE hospitales (  
    nombre VARCHAR(50),  
    CONSTRAINT pk_hospitales PRIMARY KEY (nombre)  
);
```

```
CREATE TABLE salas (  
    nombre VARCHAR(50),  
    numero INT,  
    camas SMALLINT,  
    CONSTRAINT pk_salas PRIMARY KEY (nombre, numero),  
    CONSTRAINT fk_salas FOREIGN KEY (nombre) REFERENCES  
hospitales(nombre) ON DELETE CASCADE ON UPDATE CASCADE  
);
```

```
CREATE TABLE pacientes (  
    dni INT,  
    nombre_apellido VARCHAR(50),  
    fecha_nacimiento DATE,  
    edad INT,  
    CONSTRAINT pk_pacientes PRIMARY KEY (dni)  
);
```

```
CREATE TABLE acuden_a (  
    nombre VARCHAR(50),  
    dni INT,  
    CONSTRAINT pk_acuden_a PRIMARY KEY (nombre, dni),
```

```
    CONSTRAINT fk_acuden_a_hospitales FOREIGN KEY (nombre) REFERENCES
hospitales(nombre),
    CONSTRAINT fk_acuden_a_pacientes FOREIGN KEY (dni) REFERENCES
pacientes(dni)
);
```

```
CREATE TABLE obras_sociales (
    nombre VARCHAR(50),
    CONSTRAINT pk_obras_sociales PRIMARY KEY (nombre)
);
```

```
ALTER TABLE personal
ADD COLUMN nombre_hospital VARCHAR(50),
ADD CONSTRAINT fk_personal FOREIGN KEY (nombre_hospital) REFERENCES
hospitales(nombre);
```

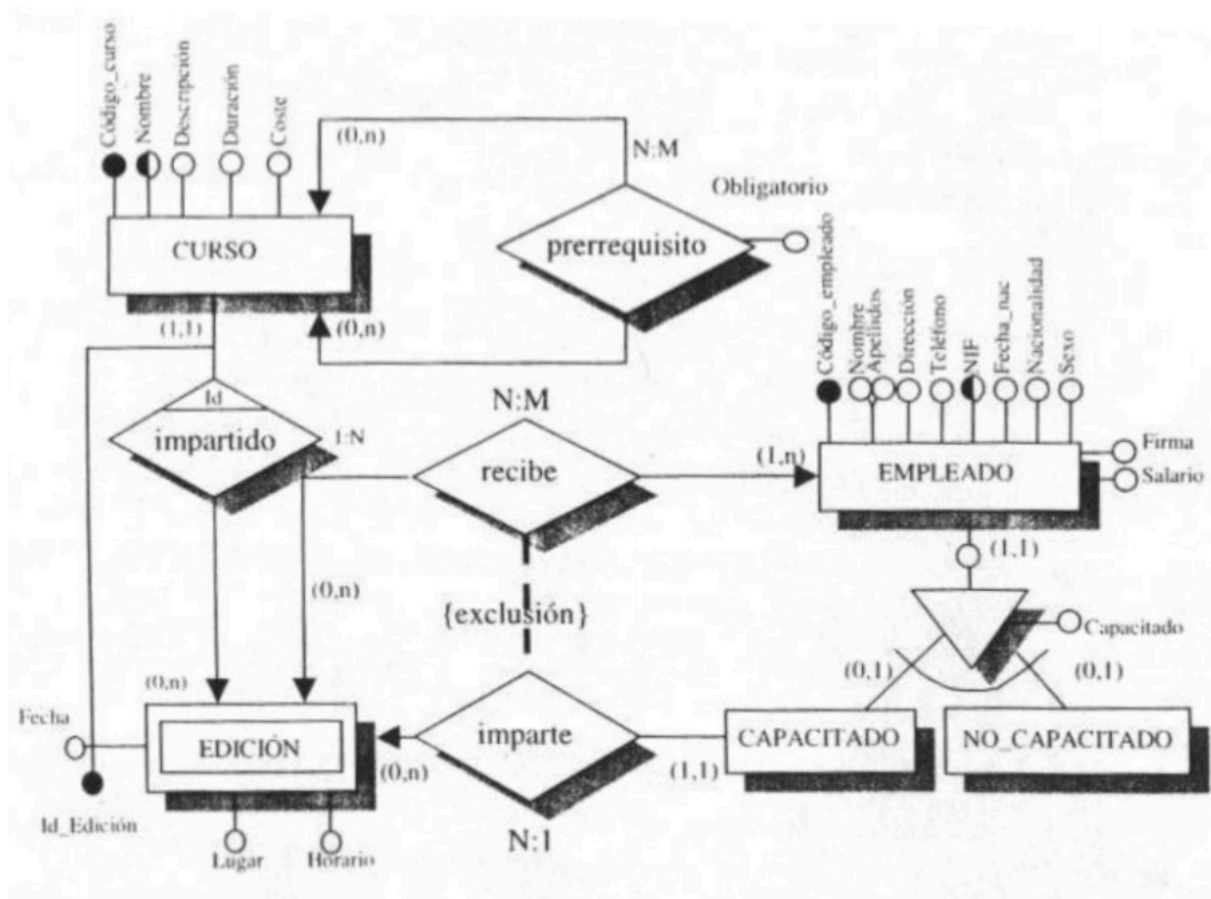
```
ALTER TABLE pacientes
ADD COLUMN nombreobra_social VARCHAR(50),
ADD COLUMN numero INT NOT NULL,
ADD CONSTRAINT fk_pacienteobra_social FOREIGN KEY (nombreobra_social)
REFERENCES obras_sociales(nombre);
```

```
ALTER TABLE tratamientos
ADD COLUMN nombre_enfermedad VARCHAR(50) NOT NULL,
ADD CONSTRAINT fk_trat_enf FOREIGN KEY (nombre_enfermedad) REFERENCES
enfermedades(nombre),
ADD COLUMN dni_med INT NOT NULL,
ADD CONSTRAINT fk_trat_dni FOREIGN KEY (dni_med) REFERENCES medicos(dni),
ADD COLUMN id_internacion INT NOT NULL,
ADD CONSTRAINT fk_trat_id FOREIGN KEY (id_internacion) REFERENCES
internaciones(id);
```

```
ALTER TABLE internaciones
ADD COLUMN numero_sala INT NOT NULL,
ADD COLUMN nombre_sala VARCHAR(50) NOT NULL,
ADD COLUMN fecha DATE NOT NULL,
ADD COLUMN nombre_hospital VARCHAR(50) NOT NULL,
ADD CONSTRAINT fk_num_sala FOREIGN KEY (nombre_sala, numero_sala)
REFERENCES salas(nombre, numero),
ADD CONSTRAINT fk_int_nomhosp FOREIGN KEY (nombre_hospital) REFERENCES
hospitales(nombre),
ADD COLUMN dni_paciente INT NOT NULL,
ADD CONSTRAINT fk_int_dnipac FOREIGN KEY (dni_paciente) REFERENCES
pacientes(dni);
```

## Ejercicio 2: Cursos de formación

Dado el siguiente enunciado y DER resultante:



El departamento de formación de una empresa desea construir una base de datos para planificar y gestionar la formación de sus empleados.

La empresa organiza cursos internos de formación de los que se desea conocer el código de curso, el nombre, una descripción, el número de horas de duración y el coste del curso.

Un curso puede tener como prerrequisito haber realizado otro(s) previamente, y a su vez la realización de un curso puede ser prerrequisito de otros. Un curso que es un prerrequisito de otro puede serlo de forma obligatoria o sólo recomendable.

Un mismo curso tiene diferentes ediciones, es decir, se imparte en diferentes lugares, fechas y con diferentes horarios (intensivo, de mañana o de tarde). En una misma fecha de inicio sólo puede impartirse una edición de un curso.

Los cursos se imparten por personal de la propia empresa.

De los empleados se desea almacenar su código de empleado, nombre y apellidos, dirección, teléfono, NIF, fecha de nacimiento, nacionalidad, sexo, firma y salario, así como si está o no capacitado para impartir cursos.

Un mismo empleado puede ser docente en una edición de un curso y alumno en otra edición, pero nunca puede ser ambas cosas a la vez (en una misma edición de curso o lo imparte o lo recibe).

### Punto 1

Describir el proceso realizado para pasar al esquema relacional de la base de datos correspondiente, detallando decisiones de diseño tomadas durante el mismo. Si hay elementos que aún no sabe cómo o no tiene herramientas para implementarlos (p.e. algunas restricciones), escríbalos.

### Punto 2

Curso				
NOMBRE DE LA COLUMNA	TIPO COLUMNA	ATRIBUTOS	VALOR PREDETERMINADO	DESCRIPCIÓN/COMENTARIOS
codigo_curso	INT	PRIMARY KEY	NULL	Código identificador de curso.
nombre	VARCHAR	-	NULL	Nombre del curso.
descripcion	TEXT	-	NULL	Descripción del curso.
duracion	INT	-	NULL	Duración del curso
coste	DECIMAL	-	NULL	Campo descriptivo del costo del curso
Prerrequisito				
curso_id	INT	PRIMARY KEY	NULL	Código identificador de curso.
prerrequisito_id	INT	PRIMARY KEY	NULL	Código identificador de prerrequisito.
obligatorio	BOOLEAN	-	NULL	Campo booleano detallando si un curso es obligatorio.
Empleado				
codigo_empleado	INT	PRIMARY KEY	NULL	Código identificador del empleado
nombre	VARCHAR	-	NULL	Nombre del empleado
apellido	VARCHAR	-	NULL	Apellido del empleado
direccion	VARCHAR	-	NULL	Dirección del empleado
telefono	VARCHAR	-	NULL	Teléfono del empleado.

	R			
nif	VARCHAR	-	NULL	Número de identificación fiscal del empleado.
fecha_nacimiento	DATE	-	NULL	Fecha de nacimiento del empleado.
nacionalidad	VARCHAR	-	NULL	Nacionalidad del empleado.
sexo	CHAR	-	NULL	Sexo del empleado.
firma	BLOB	-	NULL	Firma del empleado.
salario	DECIMAL	-	NULL	Salario del empleado.
Edicion				
curso_id	INT	PK FK	NULL	Código identificador del curso.
fecha_inicio	INT	-	NULL	Fecha de inicio de la edición del curso.
lugar	VARCHAR	-	NULL	Lugar en el que ocurre el curso.
horario	VARCHAR	-	NULL	Horario en el que ocurre el curso.
Imparte				
id_edicion	INT	PK FK	NULL	Código identificador de edicion.
codigo_empleado	INT	PK FK	NULL	Código identificador de empleado.
Recibe				
id_edicion	INT	PK FK	NULL	Código identificador de edicion.
codigo_empleado	INT	PK FK	NULL	Código identificador de empleado.
Capacitado				
codigo_empleado	INT	PK	NULL	Código identificador de empleado.
No Capacitado				
codigo_empleado	INT	PK FK	NULL	Código identificador de empleado.

### Punto 3

Escribir el script SQL con las sentencias de creación de los objetos en el motor de base de datos y mostrar los resultados de su ejecución.

#### Script SQL

```
DROP DATABASE IF EXISTS cursos;
CREATE DATABASE cursos;
USE cursos;
```

```
CREATE TABLE Curso (
    codigo_curso INT PRIMARY KEY,
    nombre VARCHAR(255) NOT NULL,
    descripcion TEXT,
    duracion INT NOT NULL,
    coste DECIMAL(10, 2) NOT NULL
);
```

```
CREATE TABLE Prerrequisito (
    curso_id INT,
    prerequisito_id INT,
    obligatorio BOOLEAN,
    PRIMARY KEY (curso_id, prerequisito_id),
    FOREIGN KEY (curso_id) REFERENCES Curso(codigo_curso) ON DELETE
    CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (prerequisito_id) REFERENCES Curso(codigo_curso) ON
    DELETE CASCADE ON UPDATE CASCADE
);
```

```
CREATE TABLE Empleado (
    codigo_empleado INT PRIMARY KEY,
    nombre VARCHAR(255) NOT NULL,
    apellidos VARCHAR(255) NOT NULL,
    direccion VARCHAR(255),
    telefono VARCHAR(20),
    nif VARCHAR(20) UNIQUE,
    fecha_nacimiento DATE,
    nacionalidad VARCHAR(50),
    sexo CHAR(1),
    firma BLOB,
    salario DECIMAL(10, 2) NOT NULL
);
```

```
CREATE TABLE Edicion (
    id_edicion INT PRIMARY KEY,
    curso_id INT,
```

```

        fecha_inicio DATE,
        lugar VARCHAR(255),
        horario VARCHAR(50),
        FOREIGN KEY (curso_id) REFERENCES Curso(codigo_curso) ON DELETE
CASCADE ON UPDATE CASCADE
);

CREATE TABLE Imparte (
    id_edicion INT,
    codigo_empleado INT,
    PRIMARY KEY (id_edicion, codigo_empleado),
    FOREIGN KEY (id_edicion) REFERENCES Edicion(id_edicion) ON DELETE
CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (codigo_empleado) REFERENCES Empleado(codigo_empleado)
ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE Recibe (
    id_edicion INT,
    codigo_empleado INT,
    PRIMARY KEY (id_edicion, codigo_empleado),
    FOREIGN KEY (id_edicion) REFERENCES Edicion(id_edicion) ON DELETE
CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (codigo_empleado) REFERENCES Empleado(codigo_empleado)
ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE Capacitado (
    codigo_empleado INT PRIMARY KEY,
    FOREIGN KEY (codigo_empleado) REFERENCES Empleado(codigo_empleado)
ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE No_Capacitado (
    codigo_empleado INT PRIMARY KEY,
    FOREIGN KEY (codigo_empleado) REFERENCES Empleado(codigo_empleado)
ON DELETE CASCADE ON UPDATE CASCADE
);

-- Trigger para asegurar que un empleado no pueda ser docente y alumno en
la misma edición
DELIMITER $$
CREATE TRIGGER trg_no_docente_y_alumno BEFORE INSERT ON Recibe
FOR EACH ROW
BEGIN
    IF EXISTS (SELECT 1 FROM Imparte WHERE Imparte.id_edicion =

```

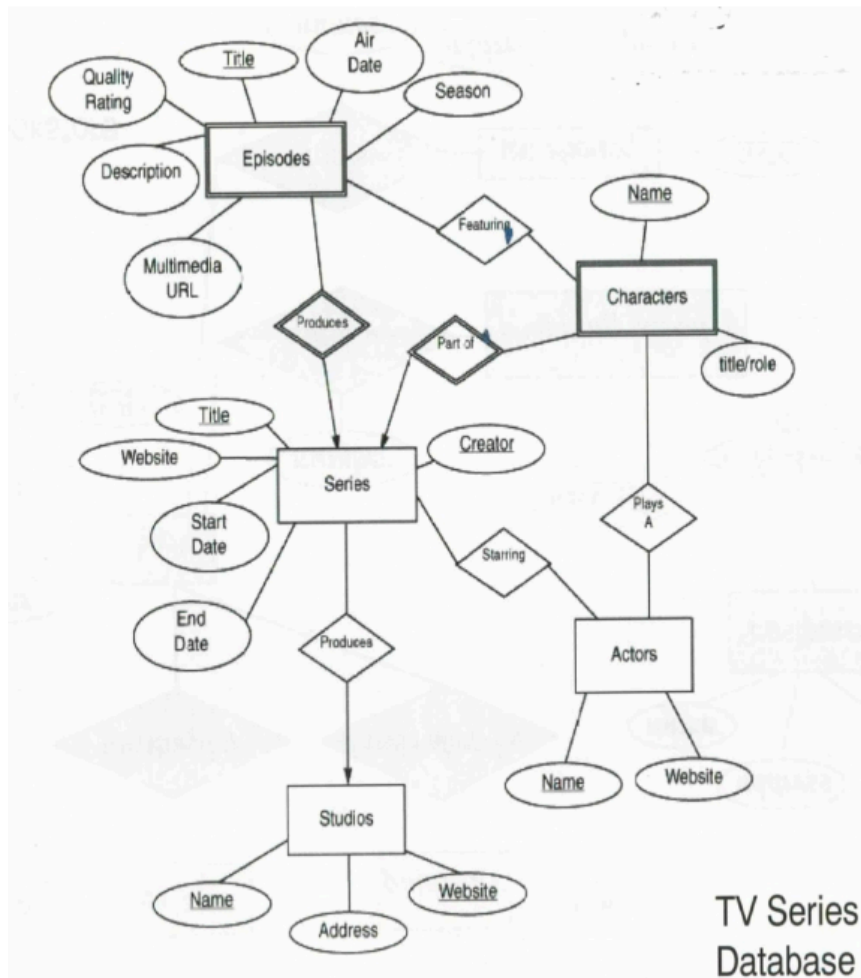
```
NEW.id_edicion AND Imparte.codigo_empleado = NEW.codigo_empleado) THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Un empleado no puede
ser docente y alumno en la misma edición.';
    END IF;
END$$
```

```
CREATE TRIGGER trg_no_alumno_y_docente BEFORE INSERT ON Imparte
FOR EACH ROW
BEGIN
    IF EXISTS (SELECT 1 FROM Recibe WHERE Recibe.id_edicion =
NEW.id_edicion AND Recibe.codigo_empleado = NEW.codigo_empleado) THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Un empleado no puede
ser alumno y docente en la misma edición.';
    END IF;
END$$
DELIMITER ;
```

### Ejercicio 3: Series de TV

Dado el siguiente DER convertir el mismo a tablas y definir los scripts de creación de los objetos en el motor de base de datos.





## Scripts SQL

```
CREATE DATABASE IF NOT EXISTS tv_series;
USE tv_series;
```

```
CREATE TABLE studios (
    name VARCHAR(50),
    website VARCHAR(50),
    address VARCHAR(50) NOT NULL,
    CONSTRAINT pk_studios PRIMARY KEY(name, website)
);
```

```
CREATE TABLE actors (
    name VARCHAR(50),
    website VARCHAR(50),
    CONSTRAINT pk_actors PRIMARY KEY(name)
);
```

-- "Al no haber restricción de participación, los atributos de la clave

foránea admiten ser nulas"

```
CREATE TABLE series (  
    end_date date COMMENT "Finalización de la emisión",  
    start_date date COMMENT "Comienzo de la emisión",  
    website VARCHAR(50),  
    title VARCHAR(50),  
    creator VARCHAR(50),  
    studios_name VARCHAR(50),  
    studios_website VARCHAR(50),  
    CONSTRAINT pk_series PRIMARY KEY(title, creator),  
    CONSTRAINT fk_series FOREIGN KEY(studios_name, studios_website)  
REFERENCES studios(name, website)  
);
```

```
CREATE TABLE episodes (  
    multimedia_url VARCHAR(50),  
    description TEXT,  
    quality_rating FLOAT,  
    title VARCHAR(50),  
    air_date DATE,  
    season INT,  
    series_title VARCHAR(50),  
    series_creator VARCHAR(50),  
    CONSTRAINT pk_episodes PRIMARY KEY(title, series_title,  
series_creator),  
    CONSTRAINT fk_episodes FOREIGN KEY(series_title, series_creator)  
REFERENCES series(title, creator) ON DELETE CASCADE ON UPDATE CASCADE  
);
```

```
CREATE TABLE characters (  
    title_or_role VARCHAR(50),  
    name VARCHAR(50),  
    series_title VARCHAR(50),  
    series_creator VARCHAR(50),  
    CONSTRAINT pk_characters PRIMARY KEY(name, series_title,  
series_creator),  
    CONSTRAINT fk_characters FOREIGN KEY(series_title, series_creator)  
REFERENCES series(title, creator) ON DELETE CASCADE ON UPDATE CASCADE  
);
```

```
CREATE TABLE starring (  
    series_title VARCHAR(50),  
    series_creator VARCHAR(50),  
    actors_name VARCHAR(50),  
    CONSTRAINT pk_starring PRIMARY KEY(series_title, series_creator,  
actors_name),
```

```

        CONSTRAINT fk_starring_series FOREIGN KEY(series_title,
series_creator) REFERENCES series(title, creator),
        CONSTRAINT fk_starring_actors FOREIGN KEY(actors_name) REFERENCES
actors(name)
);

```

```

CREATE TABLE plays_a (
    actors_name VARCHAR(50),
    characters_name VARCHAR(50),
    characters_series_title VARCHAR(50),
    characters_series_creator VARCHAR(50),
    CONSTRAINT pk_plays_a PRIMARY KEY(actors_name, characters_name,
characters_series_title, characters_series_creator),
    CONSTRAINT fk_plays_a_actors FOREIGN KEY(actors_name) REFERENCES
actors(name),
    CONSTRAINT fk_plays_a_characters FOREIGN KEY(characters_name,
characters_series_title, characters_series_creator) REFERENCES
characters(name, series_title, series_creator)
);

```

```

CREATE TABLE featuring (
    episodes_title VARCHAR(50),
    episodes_series_title VARCHAR(50),
    episodes_series_creator VARCHAR(50),
    characters_name VARCHAR(50),
    characters_series_title VARCHAR(50),
    characters_series_creator VARCHAR(50),
    CONSTRAINT pk_featuring PRIMARY KEY(episodes_title,
episodes_series_title, episodes_series_creator, characters_name,
characters_series_title, characters_series_creator),
    CONSTRAINT fk_featuring_episodes FOREIGN KEY(episodes_title,
episodes_series_title, episodes_series_creator) REFERENCES
episodes(title, series_title, series_creator),
    CONSTRAINT fk_featuring_characters FOREIGN KEY(characters_name,
characters_series_title, characters_series_creator) REFERENCES
characters(name, series_title, series_creator)
);

```

#### Ejercicio 4: Alquiler de películas

Usando la base de datos creada en el trabajo práctico n.º1 (BDA2023) y con el escenario anterior y el modelo de datos disponible en el archivo "sakila.pdf" crea las tablas de la base

de datos “en castellano”, es decir con los nombres de tablas y campos en nuestro idioma. Realice las siguientes tareas utilizando los criterios dados:

**1.1.** Crear la tabla cliente (customer) y definir su clave principal en la misma instrucción de creación. Agregar un campo “fecha\_nacimiento” de tipo fecha. Asegurarse de que cuando se inserte un nuevo cliente, éste sea mayor de edad (edad mayor o igual a 18) controlando la diferencia entre la fecha actual y la de nacimiento. Agregar una columna virtual “edad” como un campo generado (“generated column”).

```
CREATE TABLE IF NOT EXISTS customer (  
    id_cliente SMALLINT UNSIGNED AUTO_INCREMENT,  
    id_tienda TINYINT UNSIGNED NOT NULL,  
    primer_nombre VARCHAR(45) NOT NULL,  
    apellido VARCHAR(45) NOT NULL,  
    correo_electrónico VARCHAR(50),  
    activo BOOLEAN NOT NULL DEFAULT TRUE,  
    fecha_creación DATETIME NOT NULL,  
    última_actualización TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE  
CURRENT_TIMESTAMP,  
    fecha_nacimiento DATE NOT NULL,  
    edad INT, -- La columna edad se actualiza mediante un trigger  
    CONSTRAINT pk_customer PRIMARY KEY (id_cliente),  
    CONSTRAINT fk_empleado_almacen FOREIGN KEY (id_almacen) REFERENCES  
almacen(id_almacen) ON DELETE CASCADE ON UPDATE CASCADE,  
    CONSTRAINT fk_empleado_direccion FOREIGN KEY (id_direccion) REFERENCES  
direccion(id_direccion) ON DELETE CASCADE ON UPDATE CASCADE;  
    CONSTRAINT chk_edad CHECK (edad > 18)  
)  
ENGINE = InnoDB  
DEFAULT CHARACTER SET = utf8  
COMMENT = 'Tabla que almacena todos los clientes.';
```

```
-- Crear un trigger que calcule la edad antes de insertar un nuevo  
registro
```

```
DELIMITER $$  
CREATE TRIGGER before_customer_insert  
BEFORE INSERT ON customer  
FOR EACH ROW  
BEGIN  
    SET NEW.edad = TIMESTAMPDIFF(YEAR, NEW.fecha_nacimiento,  
CURDATE());  
END$$  
DELIMITER ;
```

```
-- Crear un trigger que calcule la edad antes de actualizar un registro  
DELIMITER $$
```

```

CREATE TRIGGER before_customer_update
BEFORE UPDATE ON customer
FOR EACH ROW
BEGIN
    SET NEW.edad = TIMESTAMPDIFF(YEAR, NEW.fecha_nacimiento,
CURDATE());
END$$
DELIMITER ;

```

Aclaración: Si se quisiera hacer de “edad” un atributo generado, al utilizar la función CURRENT\_DATE se produciría un error, ya que esta función no es determinista, lo cual no es válido utilizar en este tipo de campos.

**1.2.** Continuar con las tablas empleado (staff), alquiler (rental), dirección (address) y país (country)

```

CREATE TABLE alquiler (
    id_alquiler INT,
    alquiler_fecha DATETIME NOT NULL,
    id_inventario MEDIUMINT NOT NULL,
    id_cliente SMALLINT NOT NULL,
    devolucion_fecha DATETIME,
    id_empleado TINYINT NOT NULL,
    ultima_actualizacion TIMESTAMP NOT NULL,
    CONSTRAINT pk_alquiler PRIMARY KEY (id_alquiler),
    CONSTRAINT fk_alquiler_cliente FOREIGN KEY (id_cliente) REFERENCES
cliente(id_cliente) ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT fk_alquiler_empleado FOREIGN KEY (id_empleado)
REFERENCES empleado(id_empleado) ON DELETE CASCADE ON UPDATE CASCADE
);

```

```

CREATE TABLE direccion(
    id_direccion SMALLINT,
    direccion VARCHAR(50) NOT NULL,
    direccion2 VARCHAR(50),
    distrito VARCHAR(20) NOT NULL,
    id_ciudad SMALLINT NOT NULL,
    codigo_postal VARCHAR(10),
    telefono VARCHAR(20) NOT NULL,
    ultima_actualizacion TIMESTAMP NOT NULL,
    CONSTRAINT pk_direccion PRIMARY KEY (id_direccion)
);

```

```

CREATE TABLE pais(
    id_pais SMALLINT,

```

```

    pais VARCHAR(50) NOT NULL,
    ultima_actualizacion TIMESTAMP NOT NULL,
    CONSTRAINT pk_pais PRIMARY KEY (id_pais)
);

```

1.3. Crear la tabla pelicula\_actor (film\_actor) con su clave principal (hacer restricción en caso de tener una CP compuesta).

```

CREATE TABLE pelicula_actor(
    id_actor SMALLINT NOT NULL,
    id_pelicula SMALLINT NOT NULL,
    ultima_actualizacion TIMESTAMP NOT NULL,
    CONSTRAINT pk_pelicula_actor PRIMARY KEY(id_actor, id_pelicula),
    CONSTRAINT fk_actor_pelicula FOREIGN KEY (id_actor) REFERENCES
actor(id_actor) ON UPDATE CASCADE,
    CONSTRAINT fk_pelicula_pelicula_actor FOREIGN KEY (id_pelicula)
REFERENCES pelicula(id_pelicula) ON UPDATE CASCADE
);

```

1.4. Crear la tabla pago (payment) con su clave principal. Hacer las correspondientes restricciones.

```

CREATE TABLE pago (
    id_pago SMALLINT,
    id_cliente SMALLINT NOT NULL,
    id_empleado TINYINT NOT NULL,
    id_alquiler INT,
    cantidad DECIMAL(5,2),
    fecha_pago DATETIME NOT NULL,
    ultima_actualizacion TIMESTAMP,
    CONSTRAINT pk_pago PRIMARY KEY (id_pago),
    CONSTRAINT fk_pago_cliente FOREIGN KEY (id_cliente) REFERENCES
cliente(id_cliente) ON UPDATE CASCADE,
    CONSTRAINT fk_pago_empleado FOREIGN KEY (id_empleado) REFERENCES
empleado(id_empleado) ON UPDATE CASCADE,
    CONSTRAINT fk_pago_alquiler FOREIGN KEY (id_alquiler) REFERENCES
alquiler(id_alquiler) ON UPDATE CASCADE
);

```

1.5. Crear la tabla almacen (store) con sus campos propios y los referenciados. Sin generar claves

```
CREATE TABLE almacen (
    id_almacen TINYINT,
    id_empleado_gerente TINYINT NOT NULL,
    id_direccion SMALLINT NOT NULL,
    ultima_actualizacion TIMESTAMP NOT NULL,
    CONSTRAINT pk_almacen PRIMARY KEY (id_almacen)
);
```

1.6. Completar el ejercicio anterior, con la creación de las claves correspondientes. Añadir a la tabla de empleado la columna sueldo\_hora.

```
ALTER TABLE almacen
ADD CONSTRAINT pk_almacen PRIMARY KEY(id_almacen),
ADD CONSTRAINT fk_gerente_almacen FOREIGN KEY (id_empleado_gerente)
REFERENCES empleado(id_empleado) ON UPDATE CASCADE ON DELETE CASCADE,
ADD CONSTRAINT fk_direccion_almacen FOREIGN KEY (id_direccion) REFERENCES
direccion(id_direccion) ON UPDATE CASCADE ON DELETE CASCADE;

ALTER TABLE empleado
ADD COLUMN sueldo_hora float;
```

1.7. Hacer que no puedan haber dos clientes con el mismo nombre y apellido.

```
ALTER TABLE cliente
ADD UNIQUE (id_direccion);
```

1.8. Crear la tabla ciudad (city) con su correspondiente CP. Agregar clave foránea ciudad\_id (city\_id) a la tabla dirección. Eliminar la columna de direcciones de la tabla cliente.

```
CREATE TABLE ciudad (
    id_ciudad SMALLINT,
    ciudad VARCHAR(50) NOT NULL,
    id_pais SMALLINT NOT NULL,
    ultima_actualizacion TIMESTAMP NOT NULL,
    CONSTRAINT pk_ciudad PRIMARY KEY (id_ciudad),
    CONSTRAINT fk_ciudad_pais FOREIGN KEY (id_pais) REFERENCES
pais(id_pais) ON DELETE CASCADE ON UPDATE CASCADE
);
```

```
ALTER TABLE direccion
ADD CONSTRAINT fk_direccion_ciudad FOREIGN KEY (id_ciudad) REFERENCES
ciudad(id_ciudad) ON DELETE CASCADE ON UPDATE CASCADE;
```

```
ALTER TABLE cliente
```

```
DROP COLUMN direcciones;
```

1.9. Agregar columnas `calle_cliente` y `altura_cliente` a la tabla `cliente`. Agregar `domicilio_empleado` (`address_id`) en tabla `empleado`.

```
ALTER TABLE cliente
ADD COLUMN calle_cliente VARCHAR(70),
ADD COLUMN altura_cliente INT;
```

```
ALTER TABLE empleado
ADD COLUMN id_direccion SMALLINT NOT NULL,
ADD CONSTRAINT fk_empleado_direccion FOREIGN KEY (id_direccion)
REFERENCES direccion(id_direccion) ON DELETE CASCADE ON UPDATE CASCADE;
```

1.10. Eliminar tabla país.

```
DROP TABLE pais;
```

1.11. Eliminar la columna `email` de la tabla `empleado`. Agregar una tabla `email_empleado` (sin clave primaria).

```
ALTER TABLE empleado
DROP COLUMN email;
```

```
CREATE TABLE email_empleado(
    id_empleado TINYINT NOT NULL,
    email VARCHAR(50),
    CONSTRAINT fk_email_empleado FOREIGN KEY (id_empleado) REFERENCES
empleado(id_empleado)
);
```

1.12. Establecer clave primaria para `email_empleado`, teniendo en cuenta que ésta corresponde a una entidad débil de empleado.

```
ALTER TABLE email_empleado
ADD CONSTRAINT pk_email_empleado PRIMARY KEY(id_empleado),
ADD CONSTRAINT fk_email_empleado FOREIGN KEY (id_empleado) REFERENCES
empleado(id_empleado) ON DELETE CASCADE;
```

1.13. Establecer que los nombres y apellidos de los empleados no tengan valores nulos.

```
ALTER TABLE empleado
MODIFY COLUMN nombre VARCHAR(45) NOT NULL,
```



```
MODIFY COLUMN apellido VARCHAR(45) NOT NULL;
```

1.14. Establecer que no se repita la calle y la altura de los clientes.

```
ALTER TABLE cliente;  
ADD UNIQUE (direccion_cliente);
```

1.15. Crear las siguientes vistas:

1.15.1. "info\_actores": con los datos de los actores y nombres de las películas en las que actuaron.

```
CREATE VIEW vista_info_actores AS  
SELECT a.first_name, a.last_name, f.title, f.description, f.release_year  
FROM actor AS a INNER JOIN film_actor AS fa ON a.actor_id = fa.actor_id  
INNER JOIN film AS f ON fa.film_id = f.film_id;
```

1.15.2. "lista\_clientes": con información básica de los clientes incluídas sus direcciones.

```
CREATE VIEW vista_lista_clientes AS  
SELECT customer_id, first_name, address  
FROM customer INNER JOIN address ON customer.address_id =  
address.address_id;
```

```
SELECT * FROM vista_lista_clientes
```

1.15.3. "info\_peliculas": con los datos de las películas y los actores que actuaron en ellas

```
CREATE VIEW vista_info_pelis AS  
SELECT a.first_name, a.last_name, f.title, f.description, f.release_year,  
f.length, f.rating, f.special_features  
FROM actor AS a INNER JOIN film_actor AS fa ON a.actor_id = fa.actor_id  
INNER JOIN film AS f ON fa.film_id = f.film_id;
```

### Ejercicio 5: Índices

7.1) Utilizando el ejercicio 5 del TP 2 (Alquiler de películas) realice las siguientes tareas:

7.1.1) Crear un índice para la tabla alquiler sobre el campo fecha\_alquiler (rental\_date).

```
CREATE INDEX índice_fecha_alquiler ON rental(rental_date);
```

Aclaración: Para “aprovechar” los índices en las consultas se deben utilizar funciones determinísticas. Por ejemplo:

```
SELECT * FROM rental WHERE rental_date BETWEEN '2005-01-01' AND  
'2005-12-31';
```

Una consulta que utilice, por ejemplo, la función YEAR() no hará uso del índice ya que esta función es no determinística. Por ejemplo:

```
SELECT * FROM rental WHERE YEAR(rental_date) = 2005;
```

Para tener detalles sobre la ejecución se puede combinar la consulta con EXPLAIN.

```
EXPLAIN SELECT * FROM rental WHERE rental_date BETWEEN '2005-01-01' AND  
'2005-12-31';
```

7.1.2) Eliminar el índice creado en el punto anterior.

```
DROP INDEX índice_fecha_alquiler ON rental;
```

7.1.3) Crear una columna localización en la tabla dirección que permita registrar coordenadas geográficas. Crear un índice que optimice la búsqueda y ordenamiento de la tabla para el campo agregado.

```
ALTER TABLE address ADD COLUMN localization VARCHAR(50) NOT NULL;  
CREATE INDEX índice_localization ON address(localization);
```

7.1.4) Crear un nuevo índice sobre la tabla empleado sobre los campos Apellido y Nombre.

```
CREATE INDEX índice_staff_first_name ON staff(first_name);  
CREATE INDEX índice_last_name ON staff(last_name);
```

7.1.5) Crear un índice que optimice la búsqueda por ciudad y dirección en la tabla direccion (índice compuesto) teniendo en cuenta que se realizan inserciones con mucha asiduidad sobre esta tabla

```
CREATE INDEX índice_dirección_ciudad ON address(city_id, address_id);
```

## Ejercicio 5: Plan de ejecución

1. Instalamos la base de datos de Sakila

```
mysql> show databases;
+-----+
| Database |
+-----+
| BDA2024  |
| information_schema |
| menagerie |
| mysql    |
| performance_schema |
| sakila   |
| sys      |
+-----+
7 rows in set (0.00 sec)
```

169	KENNETH	HOFFMAN	2006-02-15 04:34:33
170	MENA	HOPPER	2006-02-15 04:34:33
171	OLYMPIA	PFEIFFER	2006-02-15 04:34:33
172	GROUCHO	WILLIAMS	2006-02-15 04:34:33
173	ALAN	DREYFUSS	2006-02-15 04:34:33
174	MICHAEL	BENING	2006-02-15 04:34:33
175	WILLIAM	HACKMAN	2006-02-15 04:34:33
176	JON	CHASE	2006-02-15 04:34:33
177	GENE	MCKELLEN	2006-02-15 04:34:33
178	LISA	MONROE	2006-02-15 04:34:33
179	ED	GUINNESS	2006-02-15 04:34:33
180	JEFF	SILVERSTONE	2006-02-15 04:34:33
181	MATTHEW	CARREY	2006-02-15 04:34:33
182	DEBBIE	AKROYD	2006-02-15 04:34:33
183	RUSSELL	CLOSE	2006-02-15 04:34:33
184	HUMPHREY	GARLAND	2006-02-15 04:34:33
185	MICHAEL	BOLGER	2006-02-15 04:34:33
186	JULIA	ZELLWEGER	2006-02-15 04:34:33
187	RENEE	BALL	2006-02-15 04:34:33
188	ROCK	DUKAKIS	2006-02-15 04:34:33
189	CUBA	BIRCH	2006-02-15 04:34:33
190	AUDREY	BAILEY	2006-02-15 04:34:33
191	GREGORY	GOODING	2006-02-15 04:34:33
192	JOHN	SUVARI	2006-02-15 04:34:33
193	BURT	TEMPLE	2006-02-15 04:34:33
194	MERYL	ALLEN	2006-02-15 04:34:33
195	JAYNE	SILVERSTONE	2006-02-15 04:34:33
196	BELA	WALKEN	2006-02-15 04:34:33
197	REESE	WEST	2006-02-15 04:34:33
198	MARY	KEITEL	2006-02-15 04:34:33
199	JULIA	FAWCETT	2006-02-15 04:34:33
200	THORA	TEMPLE	2006-02-15 04:34:33

200 rows in set (0.00 sec)

2. Ejecute y describa qué información arroja la siguiente consulta:

```
1  SELECT f.title, c.first_name, c.last_name, a.address, ci.city, co.country
2  FROM film f, inventory i, rental r, customer c, address a, city ci, country co
3  WHERE f.film_id = i.film_id and i.inventory_id = r.inventory_id
4  and r.customer_id = c.customer_id and c.address_id = a.address_id
5  and a.city_id = ci.city_id and ci.country_id = co.country_id
6  and f.title = 'ACADEMY DINOSAUR' and co.country LIKE 'B%'
7  ORDER BY co.country, ci.city, c.last_name, c.first_name;
```

La siguiente consulta devuelve los nombres, apellidos, direcciones y ciudad de aquellos registros cuyo título de la película sea el de “ACADEMY DINOSAUR” y el país comience con la letra “B”, en este caso fueron los países Bolivia y Brasil.

	title	first_name	last_name	address	city	country
▶	ACADEMY DINOSAUR	JOEL	FRANCISCO	287 Cuautla Boulevard	Sucre	Bolivia
	ACADEMY DINOSAUR	TINA	SIMMONS	984 Effen-Alaiye Avenue	Goiânia	Brazil
	ACADEMY DINOSAUR	DEBRA	NELSON	306 Antofagasta Place	Vila Velha	Brazil

3. Ejecute la misma consulta con EXPLAIN y EXPLAIN ANALYZE y tome nota de la información mostrada. De manera similar, ejecute la consulta con MySQL Workbench, haga una captura de pantalla de la pestaña “Execution Plan” y tome nota de la información mostrada

```
mysql> EXPLAIN
-> SELECT f.title, c.first_name, c.last_name, a.address, ci.city, co.country
-> FROM film f, inventory i, rental r, customer c, address a, city ci, country co
-> WHERE f.film_id = i.film_id
-> AND i.inventory_id = r.inventory_id
-> AND r.customer_id = c.customer_id
-> AND c.address_id = a.address_id
-> AND a.city_id = ci.city_id
-> AND ci.country_id = co.country_id
-> AND f.title = 'ACADEMY DINOSAUR'
-> AND co.country LIKE 'B%'
-> ORDER BY co.country, ci.city, c.last_name, c.first_name;
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	f		ref	PRIMARY, idx_title	idx_title	514	const	1	100.00	Using index; Using temporary; Using filesort
1	SIMPLE	i		ref	PRIMARY, idx_fk_film_id	idx_fk_film_id	2	sakila.f.film_id	4	100.00	Using index
1	SIMPLE	r		ref	idx_fk_inventory_id, idx_fk_customer_id	idx_fk_inventory_id	3	sakila.i.inventory_id	3	100.00	NULL
1	SIMPLE	c		eq_ref	PRIMARY, idx_fk_address_id	PRIMARY	2	sakila.r.customer_id	1	100.00	NULL
1	SIMPLE	a		eq_ref	PRIMARY, idx_fk_city_id	PRIMARY	2	sakila.c.address_id	1	100.00	NULL
1	SIMPLE	ci		eq_ref	PRIMARY, idx_fk_country_id	PRIMARY	2	sakila.a.city_id	1	100.00	NULL
1	SIMPLE	co		eq_ref	PRIMARY	PRIMARY	2	sakila.ci.country_id	1	11.11	Using where

```
mysql> EXPLAIN ANALYZE
-> SELECT f.title, c.first_name, c.last_name, a.address, ci.city, co.country
-> FROM film f, inventory i, rental r, customer c, address a, city ci, country co
-> WHERE f.film_id = i.film_id
-> AND i.inventory_id = r.inventory_id
-> AND r.customer_id = c.customer_id
-> AND c.address_id = a.address_id
-> AND a.city_id = ci.city_id
-> AND ci.country_id = co.country_id
-> AND f.title = 'ACADEMY DINOSAUR'
-> AND co.country LIKE 'B%'
-> ORDER BY co.country, ci.city, c.last_name, c.first_name;
```

```

EXPLAIN

--> Sort: co.country, ci.city, c.last_name, c.first_name (actual time=0.656..0.656 rows=3 loops=1)
--> Stream results (cost=39.3 rows=1.86) (actual time=0.107..0.634 rows=3 loops=1)
--> Nested loop inner join (cost=20.3 rows=1.86) (actual time=0.102..0.625 rows=3 loops=1)
--> Nested loop inner join (cost=24.5 rows=16.7) (actual time=0.0923..0.536 rows=23 loops=1)
--> Nested loop inner join (cost=18.6 rows=16.7) (actual time=0.0862..0.443 rows=23 loops=1)
--> Nested loop inner join (cost=12.8 rows=16.7) (actual time=0.0783..0.34 rows=23 loops=1)
--> Nested loop inner join (cost=6.93 rows=16.7) (actual time=0.0658..0.201 rows=23 loops=1)
--> Nested loop inner join (cost=1.08 rows=4.78) (actual time=0.0376..0.0466 rows=8 loops=1)
--> Covering index lookup on f using idx_title (title='ACADEMY DINOSAUR') (cost=0.35 rows=1) (actual time=0.0239..0.0256 rows=1 loops=1)
--> Covering index lookup on i using idx_fk_film_id (film_id=f.film_id) (cost=0.729 rows=4.78) (actual time=0.0122..0.0179 rows=8 loops=1)
--> Index lookup on r using idx_fk_inventory_id (inventory_id=i.inventory_id) (cost=0.947 rows=3.5) (actual time=0.0163..0.0184 rows=2.88 loops=8)
--> Single-row index lookup on c using PRIMARY (customer_id=r.customer_id) (cost=0.256 rows=1) (actual time=0.00555..0.00562 rows=1 loops=23)
--> Single-row index lookup on a using PRIMARY (address_id=c.address_id) (cost=0.256 rows=1) (actual time=0.00398..0.00405 rows=1 loops=23)
--> Single-row index lookup on ci using PRIMARY (city_id=a.city_id) (cost=0.256 rows=1) (actual time=0.00355..0.00362 rows=1 loops=23)
--> Filter: (co.country like 'B%') (cost=0.251 rows=0.111) (actual time=0.00341..0.00345 rows=0.13 loops=23)
--> Single-row index lookup on co using PRIMARY (country_id=ci.country_id) (cost=0.251 rows=1) (actual time=0.00274..0.00282 rows=1 loops=23)

```

```

1 * EXPLAIN ANALYZE
2 SELECT f.title, c.first_name, c.last_name, a.address, ci.city, co.country
3 FROM film f, inventory i, rental r, customer c, address a, city ci, country co
4 WHERE f.film_id = i.film_id
5 AND i.inventory_id = r.inventory_id
6 AND r.customer_id = c.customer_id
7 AND c.address_id = a.address_id
8 AND a.city_id = ci.city_id
9 AND ci.country_id = co.country_id
10 AND f.title = 'ACADEMY DINOSAUR'
11 AND co.country LIKE 'B%'
12 ORDER BY co.country, ci.city, c.last_name, c.first_name;

```

< | Form Editor | Navigate: << < 1 / 1 > >> |

EXPLAIN:

```

--> Sort: co.country, ci.city, c.last_name, c.first_name (actual time=0.236..0.237 rows=3 loops=1)
--> Stream results (cost=29.3 rows=1.79) (actual time=0.0522..0.226 rows=3 loops=1)
--> Nested loop inner join (cost=29.3 rows=1.79) (actual time=0.0504..0.223 rows=3 loops=1)
--> Nested loop inner join (cost=23.6 rows=16.1) (actual time=0.0458..0.193 rows=23 loops=1)

```

La salida de explain analyze proporcionará detalles adicionales sobre el tiempo de ejecución y el número de filas examinadas.

4. En la tabla de países (Country) cree un índice por el nombre del país (country)

```

mysql> create index idx_pais on country (country)
-> ;
Query OK, 0 rows affected (0,09 sec)
Records: 0 Duplicates: 0 Warnings: 0

```

5. Realice nuevamente el punto 3 y haga un cuadro comparativo con los resultados (ejecución de la consulta antes y después de la creación del índice). ¿Qué conclusiones obtiene?

Con EXPLAIN ANALYZE (con índice):

```
mysql> EXPLAIN ANALYZE SELECT * FROM rental WHERE rental_date BETWEEN '2005-01-01' AND '2005-12-31';
+-----+
| EXPLAIN
```

```

+-----+
| -> Filter: (rental.rental_date between '2005-01-01' and '2005-12-31') (cost=1625 rows=8004) (actual time=0.116..24.3 rows=15862 loops=1)
| -> Table scan on rental (cost=1625 rows=16008) (actual time=0.113..12.9 rows=16044 loops=1)
|
+-----+
```

(sin índice)

```
mysql> EXPLAIN ANALYZE SELECT * FROM rental WHERE rental_date BETWEEN '2005-01-01' AND '2005-12-31';
+-----+
| EXPLAIN
```

```

+-----+
| -> Filter: (rental.rental_date between '2005-01-01' and '2005-12-31') (cost=1625 rows=8004) (actual time=0.149..34.3 rows=15862 loops=1)
| -> Table scan on rental (cost=1625 rows=16008) (actual time=0.144..19 rows=16044 loops=1)
|
+-----+
```

Con EXPLAIN (con índice):

```
mysql> CREATE INDEX indice_fecha_alquiler ON rental(rental_date);
Query OK, 0 rows affected (0,22 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> EXPLAIN SELECT * FROM rental WHERE rental_date BETWEEN '2005-01-01' AND '2005-12-31';
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | rental | NULL | ALL | rental_date,indice_fecha_alquiler | NULL | NULL | NULL | 16008 | 50.00 | Using where |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set, 1 warning (0,00 sec)

mysql>
```

(sin índice)

```
mysql> ALTER TABLE rental DROP INDEX indice_fecha_alquiler;
Query OK, 0 rows affected (0,06 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> EXPLAIN SELECT * FROM rental WHERE rental_date BETWEEN '2005-01-01' AND '2005-12-31';
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | rental | NULL | ALL | rental_date | NULL | NULL | NULL | 16008 | 50.00 | Using where |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set, 1 warning (0,00 sec)

mysql>
```

EXPLAIN	EXPLAIN ANALYZE
Muestra estimaciones	Tiempos de ejecución reales y filas leídas.
No ejecuta la consulta; solo te da un plan de ejecución estimado	Ejecuta la consulta y proporciona datos reales, lo que te permite ver cómo un índice afecta realmente el rendimiento.
No impacta en el rendimiento	Impacta en el rendimiento
Te dice qué índices son elegibles y cuál será utilizado por el plan de ejecución.	Te dice si el índice mejora el rendimiento en la práctica